The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV009-SW-65

Windows Device Driver

High Speed Sync/Async Serial Interface

Version 2.0.x

User Manual

Issue 2.0.0

May 2011

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com www.tews.com

TDRV009-SW-65

Windows Device Driver

High Speed Sync/Async Serial Interface

Supported Modules:
TPMC863
TPMC363
TAMC863
TCP863

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2007-2011 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	May 21, 2007
1.0.1	Description of missing ioctl functions added (RTS/CTS/DTR/DSR)	June 19, 2007
1.0.2	Files moved to subdirectory	June 23, 2008
2.0.0	Support for Windows 7 added, API functions and parameters modified	May 27, 2011

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Software Installation 5

2.1.1 Windows 2000 / XP 5

2.1.2 Windows 7 6

2.2 Confirming Driver Installation 6

3 API DOCUMENTATION 7

3.1 General Functions 7

3.1.1 tdrv009Open 7

3.1.2 tdrv009Close 9

3.2 Device Access Functions 11

3.2.1 tdrv009Write 11

3.2.2 tdrv009WriteOverlapped 13

3.2.3 tdrv009Read 16

3.2.4 tdrv009SetOperationMode 19

3.2.5 tdrv009GetOperationMode 27

3.2.6 tdrv009SetBaudrate 34

3.2.7 tdrv009SetReceiverState 36

3.2.8 tdrv009ClearRxBuffer 38

3.2.9 tdrv009RtsSet 40

3.2.10 tdrv009RtsClear 42

3.2.11 tdrv009CtsGet 44

3.2.12 tdrv009DtrSet 46

3.2.13 tdrv009DtrClear 48

3.2.14 tdrv009DsrGet 50

3.2.15 tdrv009SetExternalXtal 52

3.2.16 tdrv009SccRegisterRead 54

3.2.17 tdrv009SccRegisterWrite 56

3.2.18 tdrv009GlobalRegisterRead 58

3.2.19 tdrv009GlobalRegisterWrite 60

3.2.20 tdrv009EepromRead 62

3.2.21 tdrv009EepromWrite 64

3.2.22 tdrv009WaitForInterrupt 66

3.2.23 tdrv009EventRegister 68

3.2.24 tdrv009EventUnregister 70

1 Introduction

The TDRV009-SW-65 Windows device driver is a kernel mode driver which allows the operation of the supported hardware module on an Intel or Intel-compatible Windows operating system. Supported Windows versions are:

- Windows 2000
- Windows XP
- Windows XP Embedded
- Windows 7 (32bit and 64bit)

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

The TDRV009-SW-65 device driver supports the following features:

- Setup and configure a channel
- Send and receive Data Buffers
- Switch on or off a channel's receiver
- Read and write onboard registers directly
- Wait for Receive Events
- Wait for interrupt events

The TDRV009-SW-65 device driver supports the modules listed below:

TPMC863	4 Channel Interface	(PMC)
TPMC363	4 Channel Interface	(PMC, Conduction Cooled)
TAMC863	4 Channel Interface	(AMC)
TCP863	4 Channel Interface	(CompactPCI)

In this document all supported modules and devices will be called TDRV009. Specials for a certain device will be advised.

To get more information about the features and use of TDRV009 devices it is recommended to read the manuals listed below.

TPMC863 Family User Manual
TPMC863 Family Engineering Manual

2 Installation

Following files are located in directory TDRV009-SW-65 on the distribution media:

i386\	Directory containing driver files for 32bit Windows versions
amd64\	Directory containing driver files for 64bit Windows versions
installer_32bit.exe	Installation tool for 32bit systems (Windows XP or later)
installer_64bit.exe	Installation tool for 64bit systems (Windows XP or later)
tdrv009.inf	Windows installation script
tdrv009.h	Header file with IOCTL codes and structure definitions
example\tdrv009exa.c	Example application
api\tdrv009api.c	Application Programming Interface source
api\tdrv009api.h	Application Programming Interface header
TDRV009-SW-65-2.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TDRV009 Device Driver on a Windows 2000 / XP operating system.

After installing the TDRV009 card(s) and boot-up your system, Windows 2000 / XP setup will show a "***New hardware found***" dialog box.

1. The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
   Click "***Next***" button to continue.
2. In the following dialog box, choose "***Search for a suitable driver for my device***".
   Click "***Next***" button to continue.
3. Insert the TDRV009 driver media; select "***Disk Drive***" in the dialog box.
   Click "***Next***" button to continue.
4. Now the driver wizard should find a suitable device driver on the media.
   Click "***Next***" button to continue.
5. Complete the upgrade device driver and click "***Finish***" to take all the changes effect.
6. Now copy all needed files (tdrv009.h and API files) to the desired target directories.

After successful installation the TDRV009 device driver will start immediately and creates devices (TDRV009_1, TDRV009_2 ...) for all recognized TDRV009 modules.

2.1.2 Windows 7

This section describes how to install the TDRV009-SW-65 Device Driver on a Windows 7 (32bit or 64bit) operating system.

Depending on the operating system type, execute the installer binaries for either 32bit or 64bit systems. This will install all required driver files using an installation wizard.

Copy needed files (tdrv009.h and API files) to desired target directory.

After successful installation a device is created for each module found (TDRV009_1, TDRV009_2 ...).

2.2 Confirming Driver Installation

To confirm that the driver has been properly loaded, perform the following steps:

1. Open the Windows Device Manager:

   a. For Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***" and click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.

   b. For Windows 7, open the "***Control Panel***" from "***My Computer***" and then click the "***Device Manager***" entry.

2. Click the "**+**" in front of "***Embedded I/O***".
   The driver "***TEWS TECHNOLOGIES – TDRV009 High Speed Serial (<ModuleName>)***" should appear for each installed device.

3 API Documentation

3.1 General Functions

3.1.1 tdrv009Open

NAME

tdrv009Open – Opens a Device

SYNOPSIS

```
TDRV009_HANDLE tdrv009Open
(
     char *DeviceName,
     int  Channel
);
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device.

Channel

This parameter describes the channel of the specified device to be used for subsequent I/O. Valid values are 0 to 3.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE hdl;

/*
** open file descriptor to device, use first channel
*/
hdl = tdrv009Open(“\\\\.\\TDRV009_1”, 0 );
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. To get extended error information, call ***GetLastError***.

ERROR CODES

All error codes are standard error codes set by the I/O system.

3.1.2 tdrv009Close

NAME

tdrv009Close – Closes a Device

SYNOPSIS

```
TDRV009_STATUS tdrv009Close
(
     TDRV009_HANDLE hdl
);
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE hdl;
TDRV009_STATUS result;

/*
** close file descriptor to device
*/
result = tdrv009Close( hdl );
if (result != TDRV009_OK)
{
     /* handle close error */
}
```

RETURNS

On success TDRV009_OK, or an appropriate error code.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2 Device Access Functions

3.2.1 tdrv009Write

NAME

tdrv009Write – Write data from a buffer to a specified device

SYNOPSIS

```
TDRV009_STATUS tdrv009Write
(
     TDRV009_HANDLE     hdl,
     char               *pData,
     int                nBytes
);
```

DESCRIPTION

This function transmits one data buffer by adding it to the DMA queue of the device. The function blocks until the data is completely transferred into the hardware FIFO. The supplied data buffer must persist until the function succeeds.

PARAMETERS

hdl

This value specifies the device handle to the hardware channel retrieved by a call to the corresponding open-function.

pData

This argument points to a user supplied buffer. The data of the buffer will be transferred to the device using Direct Memory Access.

nBytes

This parameter specifies the number of bytes to write.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;
char            *pData;

/*
** Send data on TDRV009 channel
*/
pData = (char*)malloc( 12 );
sprintf( (char*)pData, “Hello World” );

result = tdrv009Write (
              hdl,
              pData,
              strlen(pData)
         );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.2 tdrv009WriteOverlapped

NAME

tdrv009WriteOverlapped – Write data from a buffer to a specified device

SYNOPSIS

```
TDRV009_STATUS tdrv009WriteOverlapped
(
     TDRV009_HANDLE     hdl,
     char               *pData,
     int                nBytes,
     OVERLAPPED         *pOverlapped
);
```

DESCRIPTION

This function transmits one data buffer by adding it to the DMA queue of the device. An OVERLAPPED structure can be passed to this function to use overlapped I/O, causing the function to return immediately. The supplied data buffer must persist until the function succeeds.

PARAMETERS

hdl

This value specifies the device handle to the hardware channel retrieved by a call to the corresponding open-function.

pData

This argument points to a user supplied buffer. The data of the buffer will be written to the device. The data buffer must be physically contiguous and accessible by the DMA controller.

nBytes

This parameter specifies the number of bytes to write.

pOverlapped

This parameter points to a user-supplied OVERLAPPED structure. This structure must be initialized for proper overlapped I/O. Specifying NULL causes the function to block.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE   hdl;
TDRV009_STATUS   result;
char             *pData;
OVERLAPPED       OverlappedStructure;

/*
** Send data on TDRV009 channel, use overlapped I/O
*/
/* initialize Overlapped structure */
OverlappedStructure.Offset = 0;
OverlappedStructure.hEvent =  CreateEvent(
                        NULL,    // lpEventAttributes
                        TRUE,    // bManualReset
                        FALSE,   // bInitialState
                        NULL     // lpName
                        );
/* allocate and initialize memory for transfer */
pData = (char*)malloc( 12 );
sprintf( (char*)pData, “Hello World” );

result = tdrv009WriteOverlapped (
               hdl,
               pData,
               strlen(pData),
               &OverlappedStructure
          );
if (result == TDRV009_ERR_IOPENDING)
{
     /* Function call OK, wait on the Overlapped Event for completion */
     WaitForSingleObject(OverlappedStructure.hEvent, INFINITE );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_ERR_IOPENDING is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.3 tdrv009Read

NAME

tdrv009Read – Read Data from Device

SYNOPSIS

```
TDRV009_STATUS tdrv009Read
(
    TDRV009_HANDLE     hdl,
    TDRV009_RX_BUFFER  *pRxBuffer
);
```

DESCRIPTION

This function reads data from the receive buffer of the device. The function returns immediately, even if there is no data currently available. The caller has to verify the structure member *Valid* to determine if the buffer contains data.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRxBuffer

This value is a pointer to a TDRV009_RX_BUFFER data buffer. The received data is copied into this user-supplied buffer by the driver.

```
typedef struct {
    ULONG    NumberOfBytes;
    ULONG    Valid;
    ULONG    Overflow;
    UCHAR    pData[1];    /* dynamically expandable */
} TDRV009_RX_BUFFER;
```

NumberOfBytes

Returns the amount of valid bytes inside the buffer.

Valid

This OR'ed value describes if the returned buffer contains valid data. Additionally, the FrameEnd status is returned. This value consists of the following OR'ed values (defined in *tdrv009.h*):

Value	Description
TDRV009_RXBUF_DATAVALID	The data buffer contains valid data.
TDRV009_RXBUF_FRAMEEND	The data buffer contains a FrameEnd mark.

Overflow

This value marks an internal buffer overflow.

pData

The received values are copied into this buffer. It must be large enough to hold all data.

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE     hdl;
TDRV009_STATUS     result;
TDRV009_RX_BUFFER  *pRxBuf;

/*
** read one buffer with up to 100 data bytes
*/
BufferSize = 100*sizeof(unsigned char) + sizeof(TDRV009_RX_BUFFER);
pRxBuf = (TDRV009_RX_BUFFER*)malloc( BufferSize );

result = tdrv009Read( hdl, pRxBuf );
if (result == TDRV009_OK)
{
     if (  (pRxBuf->Valid & TDRV009_RXBUF_DATAVALID) &&
           (pRxBuf->NumberOfBytes > 0) ) {
          printf( "Received %d valid bytes.\n", pRxBuf->NumberOfBytes );
     }
} else {
     /* handle error */
}
free(pRxBuf);
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.4 tdrv009SetOperationMode

NAME

tdrv009SetOperationMode – Configure Channel Operation Mode

SYNOPSIS

```
TDRV009_STATUS tdrv009SetOperationMode
(
     TDRV009_HANDLE          hdl,
     TDRV009_OPERATION_MODE  *pOperationMode
);
```

DESCRIPTION

This function configures the channel's operation mode.

A call to this function must be done prior to any communication operation, because after driver startup, the channel's transceivers are disabled.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pOperationMode

This argument points to a TDRV009_OPERATION_MODE_STRUCT structure. It is necessary to completely initialize the structure. This can be done by calling the API function tdrv009GetOperationMode described below.

```
typedef struct
{
   TDRV009_COMM_TYPE              CommType;
   TDRV009_TRANSCEIVER_MODE       TransceiverMode;
   TDRV009_ENABLE_DISABLE         Oversampling;
   TDRV009_BRGSOURCE              BrgSource;
   TDRV009_TXCSOURCE              TxClkSource;
   unsigned long                  TxClkOutput;
   TDRV009_RXCSOURCE              RxClkSource;
   TDRV009_CLKMULTIPLIER          ClockMultiplier;
   unsigned long                  Baudrate;
   unsigned char                  ClockInversion;
   unsigned char                  Encoding;
   TDRV009_PARITY                 Parity;
   int                            Stopbits;
   int                            Databits;
   TDRV009_ENABLE_DISABLE         UseTermChar;
   char                           TermChar;
   TDRV009_ENABLE_DISABLE         HwHs;
   TDRV009_CRC                    Crc;
} TDRV009_OPERATION_MODE_STRUCT;
```

CommType

This parameter describes the general communication type for the specific channel. Possible values are:

Value	Description
TDRV009_COMMTYPE_ASYNC	Asynchronous communication
TDRV009_COMMTYPE_HDLC_ADDR0	Standard HDLC communication without address recognition. Used for synchronous communication.
TDRV009_COMMTYPE_HDLC_TRANSP	Extended Transparent mode. No protocol processing, channel works as simple bit collector.

TransceiverMode

This parameter describes the transceiver mode of the programmable multi-protocol transceivers. Possible values are:

Value	Description
TDRV009_TRNSCVR_NOT_USED	Default V.11
TDRV009_TRNSCVR_RS530A	EIA-530A (V.11 / V.10)
TDRV009_TRNSCVR_RS530	EIA-530 (V.11), also suitable for RS422
TDRV009_TRNSCVR_X21	X.21 (V.11)
TDRV009_TRNSCVR_V35	V.35 (V.35 / V.28)
TDRV009_TRNSCVR_RS449	EIA-449 (V.11)
TDRV009_TRNSCVR_V36	V.36 (V.11)
TDRV009_TRNSCVR_RS232	EIA-232 (V.28)
TDRV009_TRNSCVR_V28	V.28 (V.28)
TDRV009_TRNSCVR_NO_CABLE	High impedance

Oversampling

This parameter enables or disables 16times oversampling, used for asynchronous communication. For communication with standard UARTs it is recommended to enable this feature. Valid values are:

Value	Description
TDRV009_DISABLED	The 16 times oversampling is not used.
TDRV009_ENABLED	The 16 times oversampling is used.

BrgSource

This parameter specifies the frequency source used as input to the BRG (Baud Rate Generator). Valid values are:

Value	Description
TDRV009_BRGSRC_XTAL1	XTAL1 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL2	XTAL2 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL3	XTAL3 oscillator is used for BRG input
TDRV009_BRGSRC_RXCEXTERN	External clock at RxC input used for BRG input
TDRV009_BRGSRC_TXCEXTERN	External clock at TxC input used for BRG input

TxClkSource

This parameter specifies the frequency source used as input to the transmit engine. Valid values are:

Value	Description
TDRV009_TXCSRC_BRG	Baud Rate Generator output used for Tx clock
TDRV009_TXCSRC_BRGDIV16	BRG output divided by 16 used for Tx clock
TDRV009_TXCSRC_RXCEXTERN	External clock at RxC input used for Tx clock
TDRV009_TXCSRC_TXCEXTERN	External clock at TxC input used for Tx clock
TDRV009_TXCSRC_DPLL	DPLL output used for Tx clock

TxClkOutput

This parameter specifies which output lines are used to output the transmit clock, e.g. for synchronous communication. The given values can be binary OR'ed. Valid values are:

Value	Description
TDRV009_TXCOUT_TXC	Transmit clock available at TxC output line
TDRV009_TXCOUT_RTS	Transmit clock available at RTS output line

RxClkSource

This parameter specifies the frequency source used as input to the receive engine. Valid values are:

Value	Description
TDRV009_RXCSRC_BRG	Baud Rate Generator output used for Rx clock
TDRV009_RXCSRC_RXCEXTERN	External clock at RxC input used for Rx clock
TDRV009_RXCSRC_DPLL	DPLL output used for Rx clock

ClockMultiplier

This parameter specifies the multiplier used for BRG clock input. Valid values are:

Value	Description
TDRV009_CLKMULT_X1	Clock multiplier disabled
TDRV009_CLKMULT_X4	Selected input clock is multiplied by 4

Baudrate

This parameter specifies the desired frequency to be generated by the Baud Rate Generator (BRG), which can be used as clock input signal. The value is derived from the selected clocksource. Please note that only specific values depending on the selected oscillator are valid. This frequency is internally multiplied by 16, if oversampling shall be used.

ClockInversion

This parameter specifies the inversion of the transmit clock and/or the receive clock. This value can be binary OR'ed. Possible values are:

Value	Description
TDRV009_CLKINV_NONE	no clock inversion
TDRV009_CLKINV_TXC	transmit clock is inverted
TDRV009_CLKINV_RXC	receive clock is inverted

Encoding

This parameter specifies the data encoding used for communication. Valid values are:

Value	Description
TDRV009_ENC_NRZ	NRZ data encoding
TDRV009_ENC_NRZI	NRZI data encoding
TDRV009_ENC_FM0	FM0 data encoding
TDRV009_ENC_FM1	FM1 data encoding
TDRV009_ENC_MANCHESTER	Manchester data encoding

Parity

This parameter specifies the parity bit generation used for asynchronous communication. Valid values are:

Value	Description
TDRV009_PAR_DISABLED	No parity generation is used.
TDRV009_PAR_EVEN	EVEN parity bit
TDRV009_PAR_ODD	ODD parity bit
TDRV009_PAR_SPACE	SPACE parity bit (always insert '0')
TDRV009_PAR_MARK	MARK parity bit (always insert '1')

Stopbits

This parameter specifies the number of stop bits to use for asynchronous communication. Possible values are 1 or 2.

Databits

This parameter specifies the number of data bits to use for asynchronous communication. Possible values are 5 to 8.

UseTermChar

This parameter enables or disables the usage of a termination character for asynchronous communication. Valid values are:

Value	Description
TDRV009_DISABLED	A termination character is not used.
TDRV009_ENABLED	A termination character is used.

TermChar

This parameter specifies the termination character. After receiving this termination character, the communication controller will forward the received data packet immediately to the host system and use a new data packet for further received data. Any 8bit value may be used for this parameter.

HwHs

This parameter enables or disables the hardware handshaking mechanism using RTS/CTS. Valid values are:

Value	Description
TDRV009_DISABLED	Hardware handshaking is not used.
TDRV009_ENABLED	Hardware handshaking is used.

Crc

This parameter is a structure describing the CRC checking configuration.

```
typedef struct
{
   TDRV009_CRC_TYPE           Type;
   TDRV009_ENABLE_DISABLE     RxChecking;
   TDRV009_ENABLE_DISABLE     TxGeneration;
   TDRV009_CRC_RESET          ResetValue;
} TDRV009_CRC;
```

Type

This parameter describes the CRC type to be used. Possible values are:

Value	Description
TDRV009_CRC_16	16bit CRC algorithm is used for checksum
TDRV009_CRC_32	32bit CRC algorithm is used for checksum

RxChecking

This parameter enables or disables the receive CRC checking. Possible values are:

Value	Description
TDRV009_DISABLED	CRC checking will not be used
TDRV009_ENABLED	CRC checking will be used

TxGeneration

This parameter enables or disables the transmit CRC generation. Possible values are:

Value	Description
TDRV009_DISABLED	A CRC checksum will be generated
TDRV009_ENABLED	A CRC checksum will not be generated

ResetValue

This parameter describes the reset value for the CRC algorithm. Possible values are:

Value	Description
TDRV009_CRC_RST_FFFF	CRC reset value will be 0xFFFF
TDRV009_CRC_RST_0000	CRC reset value will be 0x0000

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;
TDRV009_OPERATION_MODE_STRUCT    OperationMode;

/*-------------------------------------------------
  Configure channel for Async / RS232 / 115200bps
  -------------------------------------------------*/
OperationMode.CommType           = TDRV009_COMMTYPE_ASYNC;
OperationMode.TransceiverMode    = TDRV009_TRNSCVR _RS232;
OperationMode.Oversampling       = TDRV009_ENABLED;
OperationMode.BrgSource          = TDRV009_BRGSRC_XTAL1;
OperationMode.TxClkSource        = TDRV009_TXCSRC_BRG;
OperationMode.TxClkOutput        = 0;
OperationMode.RxClkSource        = TDRV009_RXCSRC_BRG;
OperationMode.ClockMultiplier    = TDRV009_CLKMULT_X1;
OperationMode.Baudrate           = 115200;
OperationMode.ClockInversion     = TDRV009_CLKINV_NONE;
OperationMode.Encoding           = TDRV009_ENC_NRZ;
OperationMode.Parity             = TDRV009_PAR_DISABLED;
OperationMode.Stopbits           = 1;
OperationMode.Databits           = 8;
OperationMode.UseTermChar        = TDRV009_DISABLED;
OperationMode.TermChar           = 0;
OperationMode.HwHs               = TDRV009_DISABLED;
OperationMode.Crc.Type           = TDRV009_CRC_16;
OperationMode.Crc.RxChecking     = TDRV009_DISABLED;
OperationMode.Crc.TxGeneration   = TDRV009_DISABLED;
OperationMode.Crc.ResetValue     = TDRV009_CRC_RST_FFFF;

result = tdrv009SetOperationMode(hdl, &OperationMode);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. A parameter inside the structure is invalid.

Other returned error codes are system error conditions.

3.2.5 tdrv009GetOperationMode

NAME

tdrv009GetOperationMode – Return Channel's current Operation Mode Configuration

SYNOPSIS

```
TDRV009_STATUS tdrv009GetOperationMode
(
     TDRV009_HANDLE          hdl,
     TDRV009_OPERATION_MODE  *pOperationMode
);
```

DESCRIPTION

This function configures the channel's operation mode.

A call to this function must be done prior to any communication operation, because after driver startup, the channel's transceivers are disabled.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pOperationMode

This argument points to a TDRV009_OPERATION_MODE_STRUCT structure.

```
typedef struct
{
   TDRV009_COMM_TYPE              CommType;
   TDRV009_TRANSCEIVER_MODE       TransceiverMode;
   TDRV009_ENABLE_DISABLE         Oversampling;
   TDRV009_BRGSOURCE              BrgSource;
   TDRV009_TXCSOURCE              TxClkSource;
   unsigned long                  TxClkOutput;
   TDRV009_RXCSOURCE              RxClkSource;
   TDRV009_CLKMULTIPLIER          ClockMultiplier;
   unsigned long                  Baudrate;
   unsigned char                  ClockInversion;
   unsigned char                  Encoding;
   TDRV009_PARITY                 Parity;
   int                            Stopbits;
   int                            Databits;
   TDRV009_ENABLE_DISABLE         UseTermChar;
   char                           TermChar;
   TDRV009_ENABLE_DISABLE         HwHs;
   TDRV009_CRC                    Crc;
} TDRV009_OPERATION_MODE_STRUCT;
```

CommType

This parameter describes the general communication type for the specific channel. Possible values are:

Value	Description
TDRV009_COMMTYPE_ASYNC	Asynchronous communication
TDRV009_COMMTYPE_HDLC_ADDR0	Standard HDLC communication without address recognition. Used for synchronous communication.
TDRV009_COMMTYPE_HDLC_TRANSP	Extended Transparent mode. No protocol processing, channel works as simple bit collector.

TransceiverMode

This parameter describes the transceiver mode of the programmable multi-protocol transceivers. Possible values are:

Value	Description
TDRV009_TRNSCVR_NOT_USED	Default V.11
TDRV009_TRNSCVR_RS530A	EIA-530A (V.11 / V.10)
TDRV009_TRNSCVR_RS530	EIA-530 (V.11), also suitable for RS422
TDRV009_TRNSCVR_X21	X.21 (V.11)
TDRV009_TRNSCVR_V35	V.35 (V.35 / V.28)
TDRV009_TRNSCVR_RS449	EIA-449 (V.11)
TDRV009_TRNSCVR_V36	V.36 (V.11)
TDRV009_TRNSCVR_RS232	EIA-232 (V.28)
TDRV009_TRNSCVR_V28	V.28 (V.28)
TDRV009_TRNSCVR_NO_CABLE	High impedance

Oversampling

This parameter enables or disables 16times oversampling, used for asynchronous communication. For communication with standard UARTs it is recommended to enable this feature. Valid values are:

Value	Description
TDRV009_DISABLED	The 16 times oversampling is not used.
TDRV009_ENABLED	The 16 times oversampling is used.

BrgSource

This parameter specifies the frequency source used as input to the BRG (Baud Rate Generator). Valid values are:

Value	Description
TDRV009_BRGSRC_XTAL1	XTAL1 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL2	XTAL2 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL3	XTAL3 oscillator is used for BRG input
TDRV009_BRGSRC_RXCEXTERN	External clock at RxC input used for BRG input
TDRV009_BRGSRC_TXCEXTERN	External clock at TxC input used for BRG input

TxClkSource

This parameter specifies the frequency source used as input to the transmit engine. Valid values are:

Value	Description
TDRV009_TXCSRC_BRG	Baud Rate Generator output used for Tx clock
TDRV009_TXCSRC_BRGDIV16	BRG output divided by 16 used for Tx clock
TDRV009_TXCSRC_RXCEXTERN	External clock at RxC input used for Tx clock
TDRV009_TXCSRC_TXCEXTERN	External clock at TxC input used for Tx clock
TDRV009_TXCSRC_DPLL	DPLL output used for Tx clock

TxClkOutput

This parameter specifies which output lines are used to output the transmit clock, e.g. for synchronous communication. The given values can be binary OR'ed. Valid values are:

Value	Description
TDRV009_TXCOUT_TXC	Transmit clock available at TxC output line
TDRV009_TXCOUT_RTS	Transmit clock available at RTS output line

RxClkSource

This parameter specifies the frequency source used as input to the receive engine. Valid values are:

Value	Description
TDRV009_RXCSRC_BRG	Baud Rate Generator output used for Rx clock
TDRV009_RXCSRC_RXCEXTERN	External clock at RxC input used for Rx clock
TDRV009_RXCSRC_DPLL	DPLL output used for Rx clock

ClockMultiplier

This parameter specifies the multiplier used for BRG clock input. Valid values are:

Value	Description
TDRV009_CLKMULT_X1	Clock multiplier disabled
TDRV009_CLKMULT_X4	Selected input clock is multiplied by 4

Baudrate

This parameter specifies the desired frequency to be generated by the Baud Rate Generator (BRG), which can be used as clock input signal. The value is derived from the selected clocksource. Please note that only specific values depending on the selected oscillator are valid. This frequency is internally multiplied by 16, if oversampling shall be used.

ClockInversion

This parameter specifies the inversion of the transmit clock and/or the receive clock. This value can be binary OR'ed. Possible values are:

Value	Description
TDRV009_CLKINV_NONE	no clock inversion
TDRV009_CLKINV_TXC	transmit clock is inverted
TDRV009_CLKINV_RXC	receive clock is inverted

Encoding

This parameter specifies the data encoding used for communication. Valid values are:

Value	Description
TDRV009_ENC_NRZ	NRZ data encoding
TDRV009_ENC_NRZI	NRZI data encoding
TDRV009_ENC_FM0	FM0 data encoding
TDRV009_ENC_FM1	FM1 data encoding
TDRV009_ENC_MANCHESTER	Manchester data encoding

Parity

This parameter specifies the parity bit generation used for asynchronous communication. Valid values are:

Value	Description
TDRV009_PAR_DISABLED	No parity generation is used.
TDRV009_PAR_EVEN	EVEN parity bit
TDRV009_PAR_ODD	ODD parity bit
TDRV009_PAR_SPACE	SPACE parity bit (always insert '0')
TDRV009_PAR_MARK	MARK parity bit (always insert '1')

Stopbits

This parameter specifies the number of stop bits to use for asynchronous communication. Possible values are 1 or 2.

Databits

This parameter specifies the number of data bits to use for asynchronous communication. Possible values are 5 to 8.

UseTermChar

This parameter enables or disables the usage of a termination character for asynchronous communication. Valid values are:

Value	Description
TDRV009_DISABLED	A termination character is not used.
TDRV009_ENABLED	A termination character is used.

TermChar

This parameter specifies the termination character. After receiving this termination character, the communication controller will forward the received data packet immediately to the host system and use a new data packet for further received data. Any 8bit value may be used for this parameter.

HwHs

This parameter enables or disables the hardware handshaking mechanism using RTS/CTS. Valid values are:

Value	Description
TDRV009_DISABLED	Hardware handshaking is not used.
TDRV009_ENABLED	Hardware handshaking is used.

Crc

This parameter is a structure describing the CRC checking configuration.

```
typedef struct
{
   TDRV009_CRC_TYPE           Type;
   TDRV009_ENABLE_DISABLE     RxChecking;
   TDRV009_ENABLE_DISABLE     TxGeneration;
   TDRV009_CRC_RESET          ResetValue;
} TDRV009_CRC;
```

Type

This parameter describes the CRC type to be used. Possible values are:

Value	Description
TDRV009_CRC_16	16bit CRC algorithm is used for checksum
TDRV009_CRC_32	32bit CRC algorithm is used for checksum

RxChecking

This parameter enables or disables the receive CRC checking. Possible values are:

Value	Description
TDRV009_DISABLED	CRC checking will not be used
TDRV009_ENABLED	CRC checking will be used

TxGeneration

This parameter enables or disables the transmit CRC generation. Possible values are:

Value	Description
TDRV009_DISABLED	A CRC checksum will be generated
TDRV009_ENABLED	A CRC checksum will not be generated

ResetValue

This parameter describes the reset value for the CRC algorithm. Possible values are:

Value	Description
TDRV009_CRC_RST_FFFF	CRC reset value will be 0xFFFF
TDRV009_CRC_RST_0000	CRC reset value will be 0x0000

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE                  hdl;
TDRV009_STATUS                  result;
TDRV009_OPERATION_MODE_STRUCT   OperationMode;

/*-------------------------------------------------
  Read Channel Operation Mode
  -------------------------------------------------*/
result = tdrv009GetOperationMode(hdl, &OperationMode);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.6 tdrv009SetBaudrate

NAME

tdrv009SetBaudrate – Configure Transmission Rate

SYNOPSIS

```
TDRV009_STATUS tdrv009SetBaudrate
(
     TDRV009_HANDLE  hdl,
     uint32_t        Baudrate
);
```

DESCRIPTION

This function sets up the transmission rate for the specific channel. This is done without changing the configuration set by tdrv009SetOperationMode. If async oversampling is enabled, the desired baudrate is internally multiplied by 16. It is important that this result can be derived from the selected clocksource. This function specifies the desired frequency which should be generated by the Baud Rate Generator (BRG).

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Baudrate

This parameter specifies the baudrate which should be generated by the Baud Rate Generator. Be sure that the baudrate can be derived from the previously selected clock source.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Set baudrate to 14400bps
  ------------------------------*/
result = tdrv009SetBaudrate(hdl, 14400);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The desired baudrate cannot be derived from the selected clock source.

Other returned error codes are system error conditions.

3.2.7 tdrv009SetReceiverState

NAME

tdrv009SetReceiverState – Clear single Output Lines to OFF

SYNOPSIS

```
TDRV009_STATUS tdrv009SetReceiverState
(
     TDRV009_HANDLE  hdl,
     uint32_t        ReceiverState
);
```

DESCRIPTION

This function sets the channel's receiver either to active or inactive.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

ReceiverState

This parameter defines the new state of the receiver. Possible values are:

Value	Description
TDRV009_RCVR_ON	The receiver is enabled.
TDRV009_RCVR_OFF	The receiver is disabled.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Enable the receiver
  ------------------------------*/
result = tdrv009SetReceiverState(hdl, TDRV009_RCVR_ON);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified.

Other returned error codes are system error conditions.

3.2.8 tdrv009ClearRxBuffer

NAME

tdrv009ClearRxBuffer – Discard all received data

SYNOPSIS

```
TDRV009_STATUS tdrv009ClearRxBuffer
(
     TDRV009_HANDLE  hdl
);
```

DESCRIPTION

This function removes all received data from the channel's receive buffer, and flushes the hardware FIFO as well.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include "tdrv009api.h"

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Clear receive buffer
  ------------------------------*/
result = tdrv009ClearRxBuffer( hdl );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.9 tdrv009RtsSet

NAME

tdrv009RtsSet – Assert RTS Signal

SYNOPSIS

```
TDRV009_STATUS tdrv009RtsSet
(
     TDRV009_HANDLE  hdl
);
```

DESCRIPTION

This function asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Assert RTS
  ------------------------------*/
result = tdrv009RtsSet( hdl );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	The channel is in handshake mode, so this function is not allowed.

Other returned error codes are system error conditions.

3.2.10 tdrv009RtsClear

NAME

tdrv009RtsClear – De-Assert RTS Signal

SYNOPSIS

```
TDRV009_STATUS tdrv009RtsClear
(
     TDRV009_HANDLE  hdl
);
```

DESCRIPTION

This function de-asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  De-Assert RTS
  ------------------------------*/
result = tdrv009RtsClear( hdl );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	The channel is in handshake mode, so this function is not allowed.

Other returned error codes are system error conditions.

3.2.11 tdrv009CtsGet

NAME

tdrv009CtsGet – Return status of CTS signal

SYNOPSIS

```
TDRV009_STATUS tdrv009CtsGet
(
     TDRV009_HANDLE  hdl,
     uint32_t        *pCtsState
);
```

DESCRIPTION

This function de-asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pCtsState

This parameter points to an unsigned long buffer where the status of the CTS signal will be stored. Depending on the state of CTS, either 0 (inactive) or 1 (active) is returned.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;
uint32_t        CtsStatus;

/*------------------------------
  Read CTS state
  ------------------------------*/
result = tdrv009CtsGet(hdl, &CtsStatus);
if (result == TDRV009_OK)
{
     /* OK */
     printf( “CTS = %d\n”, CtsStatus );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.12 tdrv009DtrSet

NAME

tdrv009DtrSet – Assert DTR Signal

SYNOPSIS

```
TDRV009_STATUS tdrv009DtrSet
(
     TDRV009_HANDLE  hdl
);
```

DESCRIPTION

This function sets the DTR signal line to HIGH. This function is only available for the 4th channel of a TDRV009 module.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Set DTR to HIGH (only valid for channel 3)
  ------------------------------*/
result = tdrv009DtrSet(hdl);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	This function is not supported by the specific channel.

Other returned error codes are system error conditions.

3.2.13 tdrv009DtrClear

NAME

tdrv009DtrClear – De-Assert DTR Signal

SYNOPSIS

```
TDRV009_STATUS tdrv009DtrClear
(
     TDRV009_HANDLE  hdl
);
```

DESCRIPTION

This function sets the DTR signal line to LOW. This function is only available for the 4th channel of a TDRV009 module.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Set DTR to LOW (only valid for channel 3)
  ------------------------------*/
result = tdrv009DtrClear(hdl);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	This function is not supported by the specific channel.

Other returned error codes are system error conditions.

3.2.14 tdrv009DsrGet

NAME

tdrv009DsrGet – Return status of DSR signal

SYNOPSIS

```
TDRV009_STATUS tdrv009DsrGet
(
     TDRV009_HANDLE  hdl,
     uint32_t        *pDsrState
);
```

DESCRIPTION

This function returns the current state of the DSR signal line of the specific channel. This function is only available for the 4th channel of a TDRV009 module

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pDsrState

This parameter points to an unsigned long buffer where the status of the DSR signal will be stored. Depending on the state of DSR, either 0 (inactive) or 1 (active) is returned.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;
uint32_t        DsrStatus;

/*------------------------------
  Read DSR state
  ------------------------------*/
result = tdrv009DsrGet(hdl, &DsrStatus);
if (result == TDRV009_OK)
{
     /* OK */
     printf( “DSR = %d\n”, DsrStatus);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_ACCESS_DENIED	This function is not supported by the specific channel.

Other returned error codes are system error conditions.

3.2.15 tdrv009SetExternalXtal

NAME

tdrv009SetExternalXtal – Configure externally supplied oscillator frequency

SYNOPSIS

```
TDRV009_STATUS tdrv009SetExternalXtal
(
     TDRV009_HANDLE  hdl,
     uint32_t        XtalFrequency
);
```

DESCRIPTION

This function specifies the frequency of an externally provided clock. This frequency is used for baudrate calculation, and describes the input frequency to the Baud Rate Generator (BRG). The external frequency may be supplied either at input line TxC or RxC.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

XtalFrequency

This parameter specifies the clock frequency in Hz.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE  hdl;
TDRV009_STATUS  result;

/*------------------------------
  Specify 1MHz as external clock frequency
  ------------------------------*/
result = tdrv009SetExternalXtal(hdl, 1000000);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.

Other returned error codes are system error conditions.

3.2.16 tdrv009SccRegisterRead

NAME

tdrv009SccRegisterRead – Read from Controller's SCC Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009SccRegisterRead
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
);
```

DESCRIPTION

This function reads one 32bit word from the communication controller's channel-specific SCC register space.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRegisterBuffer

This Parameter is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's channel-specific SCC register space, relative to the start of the channel's SCC register area. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication controller's channel-specific SCC register space.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Read a 32bit value (Status Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0004;
result = tdrv009SccRegisterRead ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     printf( “Value = 0x%X\n”, AddrBuf.Value );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.17 tdrv009SccRegisterWrite

NAME

tdrv009SccRegisterWrite – Write to Controller's SCC Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009SccRegisterWrite
(
    TDRV009_HANDLE          hdl,
    TDRV009_ADDR_STRUCT     *pRegisterBuffer
);
```

DESCRIPTION

This function writes one 32bit word to the communication controller's channel-specific SCC register space.

Modifying register contents may result in communication problems, system crash or other unexpected behavior.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRegisterBuffer

This Parameter is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's channel-specific SCC register space, relative to the start of the channel's SCC register area. Please refer to the hardware user manual for further information.

Value

This 32bit word will be written to the communication controller's channel-specific SCC register space.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Write a 32bit value (Termination Character Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0048;
AddrBuf.Value  = (1 << 15) | 0x42;
result = tdrv009SccRegisterWrite ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.18 tdrv009GlobalRegisterRead

NAME

tdrv009GlobalRegisterRead – Read from Controller's Global Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009GlobalRegisterRead
(
     TDRV009_HANDLE          hdl,
     TDRV009_ADDR_STRUCT     *pRegisterBuffer
);
```

DESCRIPTION

This function reads one 32bit word from the communication controller's Global Register Space.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRegisterBuffer

This Parameter is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's global register space. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication global register space.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Read a 32bit value (Version Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x00F0;
result = tdrv009GlobalRegisterRead ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     printf( “Value = 0x%X\n”, AddrBuf.Value );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.19 tdrv009GlobalRegisterWrite

NAME

tdrv009GlobalRegisterWrite – Write to controller's Global Register Space

SYNOPSIS

```
TDRV009_STATUS tdrv009GlobalRegisterWrite
(
     TDRV009_HANDLE          hdl,
     TDRV009_ADDR_STRUCT     *pRegisterBuffer
);
```

DESCRIPTION

This function writes one 32bit word to the communication controller's Global Register Space.

Modifying register contents may result in communication problems, system crash or other unexpected behavior.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pRegisterBuffer

This Parameter is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's global register space. Please refer to the hardware user manual for further information.

Value

This 32bit word will be written to the communication controller's global register space.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_ADDR_STRUCT     AddrBuf;

/*-------------------------------------------------
  Write a 32bit value (FIFO Control Register 4)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0034;
AddrBuf.Value  = 0xffffffff;
result = tdrv009GlobalRegisterWrite ( hdl, &AddrBuf );
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.20 tdrv009EepromRead

NAME

tdrv009EepromRead – Read from EEPROM

SYNOPSIS

```
TDRV009_STATUS tdrv009EepromRead
(
    TDRV009_HANDLE          hdl,
    TDRV009_EEPROM_BUFFER   *pEepromBuffer
);
```

DESCRIPTION

This function reads one 16bit word from the onboard EEPROM.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pEepromBuffer

This Parameter is a pointer to a *TDRV009_EEPROM_BUFFER* structure.

```
typedef struct
{
   unsigned long   Offset;
   unsigned long   Value;
} TDRV009_EEPROM_BUFFER;
```

Offset

This parameter specifies a 16bit word offset into the EEPROM.
Following offsets are available:

Offset	Access
00h – 5Fh	R
60h – 7Fh	R / W

Value

This parameter returns the 16bit word from the EEPROM at the given offset.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_EEPROM_BUFFER   EepromBuf;

/*-------------------------------------------------
  Read a 16bit value into the EEPROM, offset 0
  -------------------------------------------------*/
EepromBuf.Offset = 0;
result = tdrv009EepromRead( hdl, &EepromBuf);
if (result == TDRV009_OK)
{
     printf( “Value = 0x%X\n”, EepromBuf.Value );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.21 tdrv009EepromWrite

NAME

tdrv009EepromWrite – Write from EEPROM

SYNOPSIS

```
TDRV009_STATUS tdrv009EepromWrite
(
    TDRV009_HANDLE          hdl,
    TDRV009_EEPROM_BUFFER   *pEepromBuffer
);
```

DESCRIPTION

This function writes one 16bit word into the onboard EEPROM. The first part of the EEPROM is reserved for factory usage, write accesses to this area will result in an error.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pEepromBuffer

This Parameter is a pointer to a *TDRV009_EEPROM_BUFFER* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_EEPROM_BUFFER;
```

Offset

This parameter specifies a 16bit word offset into the EEPROM.
Following offsets are available:

Offset	Access
00h – 5Fh	R
60h – 7Fh	R / W

Value

This parameter specifies the 16bit word to be written into the EEPROM at the given offset.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_EEPROM_BUFFER   EepromBuf;

/*-------------------------------------------------
  Write a 16bit value into the EEPROM, offset 60h
  -------------------------------------------------*/
EepromBuf.Offset = 0x60;
EepromBuf.Value  = 0x1234;
result = tdrv009EepromWrite( hdl, &EepromBuf);
if (result == TDRV009_OK)
{
     /* OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid parameter specified. The specified offset is invalid.

Other returned error codes are system error conditions.

3.2.22 tdrv009WaitForInterrupt

NAME

tdrv009WaitForInterrupt – Wait for SCC Interrupt Event

SYNOPSIS

```
TDRV009_STATUS tdrv009WaitForInterrupt
(
    TDRV009_HANDLE          hdl,
    TDRV009_WAIT_STRUCT     *pWaitBuffer
);
```

DESCRIPTION

This function waits until a specified SCC interrupt or the timeout occurs.

PARAMETERS

handle

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pWaitBuffer

This parameter is a pointer to a *TDRV009_WAIT_STRUCT* structure.

```
typedef struct
{
   unsigned long   Interrupts;
   int             Timeout;
} TDRV009_WAIT_STRUCT;
```

Interrupts

This parameter specifies interrupt bits to wait for. If at least one interrupt occurs, the value is returned in this parameter. Please refer to the hardware user manual for further information on the possible SCC interrupt bits.

Timeout

This parameter specifies the time (in system ticks) to wait for an interrupt. If -1 is specified, the function will block indefinitely.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_WAIT_STRUCT     WaitStruct;

/*-------------------------------------------------
  Wait at least 10 seconds for a
  CTS Status Change (CSC) interrupt
  -------------------------------------------------*/
WaitStruct.Interrupts = (1 << 14);
WaitStruct.Timeout    = 10;

result = tdrv009WaitForInterrupt(hdl, &WaitStruct);
if (result == TDRV009_OK)
{
     printf( “Occurred Interrupt = 0x%X\n”, WaitStruct.Interrupts );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_BUSY	Too many concurrent wait jobs pending (max. 10)
TDRV009_ERR_TIMEOUT	Timeout. None of the specified events occurred.

Other returned error codes are system error conditions.

3.2.23 tdrv009EventRegister

NAME

tdrv009EventRegister – Register a User Event

SYNOPSIS

```
TDRV009_STATUS tdrv009EventRegister
(
    TDRV009_HANDLE          hdl,
    TDRV009_EVENT_STRUCT    *pEventBuffer
);
```

DESCRIPTION

This function registers a user event which is signaled by the device driver.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pEventBuffer

This parameter is a pointer to a *TDRV009_EVENT_STRUCT* structure.

```
typedef struct
{
  HANDLE   hEvent;
  ULONG    type;
} TDRV009_EVENT_STRUCT;
```

hEvent

This parameter specifies the event which should be signaled by the driver. This event must be created by a call to *CreateEvent()*.

type

This parameter specifies the event type. The only event type possible is TDRV009_RX_EVENT (refer to *tdrv009.h* for definition). It is signaled upon received data.

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_EVENT_STRUCT    EventStruct;

//
//  init event structure
//
EventStruct.type = TDRV009_RX_EVENT;
EventStruct.hEvent = CreateEvent(
                   NULL,    // lpEventAttributes
                   TRUE,    // bManualReset
                   FALSE,   // bInitialState
                   NULL     // lpName
                   );
result = tdrv009EventRegister ( hdl, &EventStruct );
if (result == TDRV009_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	The event is not initialized properly.
TDRV009_ERR_BUSY	An event is already registered.

Other returned error codes are system error conditions.

3.2.24 tdrv009EventUnregister

NAME

tdrv009EventUnregister – Unregister a User Event

SYNOPSIS

```
TDRV009_STATUS tdrv009EventUnregister
(
    TDRV009_HANDLE          hdl,
    TDRV009_EVENT_STRUCT    *pEventBuffer
);
```

DESCRIPTION

This function unregisters a previously registered user event.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pEventBuffer

This parameter is a pointer to a *TDRV009_EVENT_STRUCT* structure.

```
typedef struct
{
   HANDLE   hEvent;
   ULONG    type;
} TDRV009_EVENT_STRUCT;
```

hEvent

This parameter is not evaluated by the device driver for this function.

type

This parameter specifies the event type. The only event type possible is TDRV009_RX_EVENT (refer to *tdrv009.h* for definition).

EXAMPLE

```
#include “tdrv009api.h”

TDRV009_HANDLE          hdl;
TDRV009_STATUS          result;
TDRV009_EVENT_STRUCT    EventStruct;

//
//  init event structure
//
EventStruct.type = TDRV009_RX_EVENT;
result = tdrv009EventUnregister ( hdl, &EventStruct );
if (result == TDRV009_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV009_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TDRV009_ERR_INVALID_HANDLE	The specified TDRV009_HANDLE is invalid.
TDRV009_ERR_INVALID_PARAMETER	Invalid event type specified.
TDRV009_ERR_ACCESS_DENIED	No matching event registered.

Other returned error codes are system error conditions.